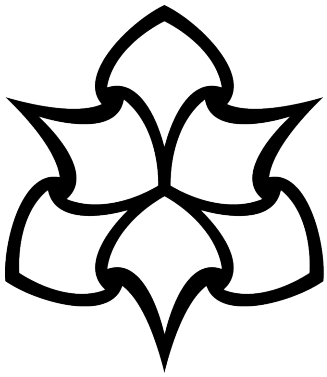


The False-Positive Rate of Automated Plagiarism Detection for SQL Assessments

Anthony Kleerekoper
a.kleerekoper@mmu.ac.uk
Manchester Metropolitan University
Manchester, UK

Andrew Schofield
a.schofield@mmu.ac.uk
Manchester Metropolitan University
Manchester, UK

ABSTRACT

Automated assessment is becoming increasingly common in Computer Science and with it automated plagiarism detection is also common. However, little attention has been paid to SQL assessment where submissions are much shorter and must be less varied than in imperative languages. This brings the challenge of avoiding high false-positive rates that require manual inspection and undermine the usefulness of automated detection.

In this paper we investigate the false-positive rate of various automated plagiarism detection algorithms. We find that there is a significant false-positive rate of between 15% and 64%. These results call into question the usefulness of automated detection for SQL since they imply that a lot of manual inspection will still be needed.

However, our results suggest that the false-positive rate may be restricted to shorter queries (e.g. under 200 characters). Further research is needed because our datasets consist mostly of short queries and the results for longer queries are based on a small subset of the data.

CCS CONCEPTS

• **Information systems → Structured Query Language**; *Data management systems*; • **Applied computing → Education** ; **Computer assisted instruction**.

KEYWORDS

plagiarism, SQL, education, automated assessment

ACM Reference Format:
Anthony Kleerekoper and Andrew Schofield. 2019. The False-Positive Rate of Automated Plagiarism Detection for SQL Assessments. In *UK & Ireland Computing Education Research Conference (UKICER), September 5–6, 2019, Canterbury, United Kingdom.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3351287.3351290

1 INTRODUCTION

Plagiarism is a significant and growing problem in Computer Science education [2, 5, 13]. A lot of research effort has been invested into trying to understand the causes of plagiarism and propose solutions [3, 6, 9, 11, 16, 17]. Much of this effort has been directed towards automatically detecting plagiarism in submitted work, with a focus on imperative languages such as C++ and Java (see [10] for a recent systematic review of such tools).

Despite the increasing use of automated and online assessment tools for SQL [1, 7, 8, 12], there has been relatively little research into detecting plagiarism in SQL queries [14, 15]. Indeed, the systematic review of automated detection tools from Novak *et al.* identifies 29 languages used in 131 articles and none of them are SQL [10].

There are two primary differences between imperative languages and SQL for the purposes of automated detection. The first is that SQL queries are much shorter than imperative code. The second is that, for a given task, there are far fewer ways to write a correct SQL answer than there are for imperative code. Both of these differences limit the amount of variability between correct answers.

With longer, imperative language assignments, the challenge for automated detection is to be effective despite the numerous ways students can hide their plagiarism. Cosma and Joy identify 51 ways in which students can hide copied code including changing variable names, reordering code, swapping data structures etc [4].

Because SQL queries are shorter and more rigidly structured, there are fewer opportunities to hide plagiarism. However, there is a different problem: false-positives. Since there are fewer ways to construct a correct answer, automated detection may flag a lot of innocent submissions as having been plagiarised. If the false-positive rate is too high then significant manual inspection is required which undermines the usefulness of the automated detection method.

Of course, a method that has a very high bar to flagging a submission as plagiarised will also be ineffective because it won't flag genuine cases and can be easily fooled. There is therefore a trade-off between effectiveness and usefulness. In this paper, we investigate that trade-off by empirically testing different detection algorithms. We compare the proportion of submissions flagged as plagiarised across two datasets whose approximate rates of plagiarism are known to us.

The first dataset, which we refer to as *exam*, is from an assessed test taken under exam conditions, where no plagiarism could have occurred. The second, called *coursework*, is from an assessed coursework where students had ample opportunity to plagiarise and where such plagiarism and collusion was observed.

An effective and useful detection method would have a very low detection rate for the exam dataset and a high rate for the coursework one. [1]. Our results cast doubt on the possibility of such a method existing. We find that using the strictest detection criteria results in a lower than expected detection rate for the coursework

UKICER, September 5–6, 2019, Canterbury, United Kingdom

ACM ISBN 978-1-4503-7257-2/19/09…$15.00
https://doi.org/10.1145/3351287.3351290

[1] By detection rate, we mean the proportion of correct submissions flagged by the program as being plagiarised, regardless of whether or not that flagging is correct

dataset while still producing a non-negligible false-positive rate in the exam dataset. Relaxing the detection method to catch more plagiarism in the coursework dataset causes a significant increase in the false-positive rate to over 50%.

However, we also found that there is a strong relationship between the length of the submitted query and the likelihood of it being flagged incorrectly as plagiarised. In the exam dataset, 60% of queries with fewer than 200 characters are flagged whereas only 5% of those with 200 or more characters are flagged. In the coursework dataset no such relationship is found and the detection rate is 80% for shorter queries (under 200 characters) and 70% for the longer ones.

This would seem to suggest that the detection algorithms are effective for longer SQL queries. But further research is needed because our datasets mostly consist of shorter queries. The average query length is 108 characters (s.d. 58) for the exam dataset and 153 (s.d 100) for the coursework one. Only 9% of the queries in the exam dataset are 200 characters or longer and so the lower detection rate may be serendipitous as a result of having fewer queries to compare to, rather than the detection algorithms being more effective with longer queries.

2 SQL PLAGIARISM DETECTION ALGORITHMS

String matching is the most common type of algorithm for automated detection [10] and the type we consider in this paper. Other methods have been proposed for longer programs where direct string comparisons may take too long or where there are too many obfuscation methods available to students. These problems do not manifest in SQL and so string matching is easily applicable.

Russell and Cumming describe an online assessment system for SQL which includes plagiarism detection [14]. They explain their motivation for wanting to apply automated detection as the desire to save time, "at least to filter out clearly non-matching submissions". Their detection tool involves three independent algorithms whose results are reported to the user. The user can then make a judgement call based on the three algorithms.

The first algorithm is called *equality* which performs simple string matching. This algorithm returns a score between 10 and 4 depending on how similar the strings are. That is, a score of 10 indicates that the two submissions match exactly without any changes being made and a score of 4 indicates that two submissions match if case, non-essential white space and brackets are ignored. They do not define the intermediate scores.

The authors noted that this algorithm alone is sufficient to "catch most students" since plagiarists attempt to hide their plagiarism by changing cases or introducing line breaks in the middle of an answer.

The second algorithm they propose is called *shuffle* and is designed to detect situations in which plagiarism is hidden by reordering of lines of code. For example, a WHERE clause with two predicates ANDed together can be made to look different by changing the order of the predicates. For the purposes of this algorithm, a line of code is defined in terms of SQL keywords such as WHERE and AND.

Name	Exam	Coursework
Plagiarism Level	None	High
Number of Submissions	753	810
Number of Students	97	63
Number of Questions	80	15
Average Submissions per Question	9	54

Table 1: A summary of the key features of the datasets used in this paper.

Other re-orderings include swapping the order of equality comparisons (i.e. `a.b = c.d` could be written as `c.d = a.b`), and joins can be put in a different order as well. *Shuffle* returns a score giving the number of lines that have to be moved around in order to make two submissions match.

Finally, a third algorithm is proposed called *histogram* which is not designed primarily as a detection method but as a way of giving evidence that plagiarism was not accidental. The algorithm looks for words or patterns that appear only in the two submissions that have been flagged as being copies of each other. For example, it may be that only these two submissions have used a particular alias, or that only these two submissions have two spaces after each operator, then that is evidence of plagiarism.

To validate these algorithms, the authors applied their university's plagiarism procedures to those students who had more than one answer flagged. All of the students that were flagged in this way were confirmed to have plagiarised through the university's procedures. As a second validation, they also applied their algorithm between cohorts and found that no cases of cross-cohort plagiarism were detected out of over 1,000 students.

Scerbakov *et al.* propose algorithms for a slightly different problem [15]. In their work, students are asked to design a database system of their choice and submit queries for their designed system. This increased flexibility makes it much easier to hide plagiarism by changing names such as table or attribute names. They therefore propose an algorithm that detects plagiarism by first replacing all names with tokens and then calculating a distance metric between the tokenised submissions.

3 EXPERIMENTAL SETUP

In this paper, we compare the performance of different detection algorithms on two datasets whose plagiarism rates are approximately known.

The first dataset, which we refer to as *exam*, is taken from an automated assessed test taken under exam conditions using the tool described by Kleerekoper and Schofield [7]. Students were not able to see the questions in advance and could not communicate with each other during the test. Each student was presented with a random set of questions drawn from a pool meaning that it is very unlikely that neighbouring students had the same questions as each other. We are confident that no plagiarism exists in this dataset.

The second dataset, which we refer to as *coursework*, is from an assessed coursework in which students were given portfolios to complete in their own time without supervision. All students were given the same questions and had many weeks to complete the work. We observed students working together or asking for help

from other students. We are confident that plagiarism occurred at a very high rate in this dataset.

A summary of the key features of the datasets is given in Table 1.

The algorithms we consider were implemented by the authors in Python using Jupyter Notebooks. For each algorithm we calculate the proportion of submissions that the algorithm flags as being plagiarised and the proportion of students with at least two answers flagged. For reproducibility the implementation is available on github though the datasets are not as they contain the answers to an assessed test used regularly[2].

4 EQUALITY ALGORITHM RESULTS

The first detection algorithm we test is the *equality* algorithm used by Russell and Cumming [14]. This algorithm uses string matching with variations. In Russel and Cumming's implementation, the algorithm returns a score between 10 and 4 depending on how closely the submissions matched. For this paper modify this slightly by dividing the variations into individual algorithms that either flag the submissions as plagiarised or not.

We consider the following variations:

(1) No relaxation (i.e. exact match as submitted)
(2) Ignore case
(3) Ignore whitespace
(4) Ignore quotation marks
(5) Ignore brackets
(6) Ignore all (case, whitespace, quotation marks and brackets)

It should be noted that line breaks are always ignored (replaced with single spaces) because, in the *exam* dataset, all line breaks are replaced with spaces by the assessment system.

The results of applying the equality method to the datasets is shown in Table 2. The Table shows the proportion of submissions that were flagged for plagiarism as well as the proportion of students who had at least two submissions flagged as being plagiarised.

The results for the coursework dataset are consistent with our belief that this dataset includes a very high rate of plagiarism. Even using the most strict comparison method, 58% of submitted answers were flagged as plagiarism and over 85% of students were flagged as having copied at least two answers. With the most relaxed variant, 77% of submissions are flagged with over 98% of students flagged for copying two or more answers. These results show that the equality algorithm is effective at identifying plagiarism.

Looking at the exam dataset, the results are cause for concern. We know that there was no (or virtually no) plagiarism in this dataset and yet over 50% of answers were flagged for plagiarism using the most relaxed detection method. Even using the very strictest method, almost 1 in 6 answers were flagged as being copied. Moreover, 50% of students were flagged as having copied at least two answers even using the strictest method and this rises to 99% with the most relaxed version. These results suggest that the false positive rate is very high and that automated detection is not useful because it cannot properly rule out cases where no plagiarism occurred.

[2] github.com/kleerekoper/SQLPlagiarismDetector

5 SHUFFLE ALGORITHM RESULTS

The second algorithm we consider in this paper is a version of the *shuffle* algorithm described by Russell and Cumming [14]. In their implementation, the shuffle algorithm returned the number of *flips* needed to make two submissions match. For our purposes, we implement the algorithm such that it flags a submission as plagiarised if it can be made equal to another by shuffling.

Because shuffle is designed to detect plagiarism even when there has been an attempt to hide it, our implementation ignores case, whitespace, quotes and brackets as well as checking for other obfuscation methods.

We consider the following variations:

(1) Removing Alias Names
(2) Swapping Equality Comparison Operands
(3) Shuffling Comparison Predicates
(4) All Variations (all of the above)

5.1 Removing Alias Names

Changing alias names (or introducing ones) is a simple way for students to try to avoid detection by string matching. In some cases, a question may not ask for an alias or leave it up to the students to choose their own and cheating students can change the alias, or leave it out or add one in to make their answer appear different. Indeed, we have observed submissions which are certainly plagiarised but where the students have changed the alias names. By removing alias names from the submissions before testing for plagiarism, we can increase the ability to detect plagiarism.

Applying this method resulted in a large increase in detection rate for the exam dateset, going from 56.5% to 63.4%. In the coursework dataset, the increase was smaller, going from 77.4% to 80.1%.

At first this result is surprising because the SQL assessment tool used for the exam dataset requires specific column names for an answer to be correct (see [7]). We would therefore expect that removing alias names would make no difference. However, upon inspection we found that students had used different ways of creating the alias (some using the AS keyword and some not), and that some students had created aliases where none were needed (e.g. `SELECT city as city`).

By contrast, in the coursework dataset, students were never asked for specific column names. Therefore far fewer submissions contained them and removing them made far less of a difference.

5.2 Swapping Equality Comparison Operands

Swapping the order of comparison operands is another simple technique to make two identical queries look different. For example, a query written with a comparison of the form `a.b = c.d` can be rewritten as `c.d = a.b` and would therefore appear different when applying string matching. Equality comparisons are the easiest types to swap because no change is required in the operands but all operators could be swapped in theory. Here we limit ourselves to equality operators because they are very common and easy to swap.

Checking for this avoidance method causes a negligible increase in the detection rate. For the exam dataset the rate increases from 56.5% to 57.3% and for the coursework dataset it is from 77.4% to 78.1%. These results show that students, at the moment at least,

	Exam Dataset (no plagiarism)		Coursework Dataset (high plagiarism)	
	Flagged Submissions	**Flagged Students**	**Flagged Submissions**	**Flagged Students**
No Relaxation	15.8%	49.3%	57.6%	87.0%
Ignore Case	34.6%	83.8%	61.9%	94.4%
Ignore Whitespace	23.0%	60.6%	69.1%	91.4%
Ignore Quotes	20.0%	56.2%	58.5%	90.1%
Ignore Brackets	16.0%	49.3%	57.9%	88.3%
Ignore All	56.5%	98.9%	77.4%	98.2%

Table 2: Proportion of submissions flagged and students with at least two submissions flagged as plagiarised using the equality algorithm.

do not regularly swap the operands of equality comparisons. This is not a surprising result because when one of the operands is a constant there is a very strong convention for the constant to be placed after the equals sign and students are unlikely to break that convention. As a way of avoiding plagiarism it is a poor tactic since, by virtue of being so unusual, it acts as a flag that something is unusual about that submission.

The other common use of the equality comparison is in join conditions. Here the convention is to refer to the first table first. For example, it would be more usual to see `FROM tblA JOIN tblB ON A.col1 = B.col2` than `FROM tblA JOIN tblB ON B.col2 = A.col1`. Therefore, when trying to detect this type of copying we would need to swap `tblA` and `tblB` as well.

We implemented this algorithm to include swapping the first two tables joined together as well as equality operands. However, there was no difference at all in the detection rates compared to just swapping the operands.

These results show that testing for swapped operands has no significant effect on the false-positive rate and is therefore likely to be safe to use in the future. On the other hand, the reason it appears safe is that it seems to have no impact on detection so may not help detect plagiarism.

5.3 Shuffling Comparison Predicates

A third method that students may be able to use with ease to avoid detection by string matching is to change the order of the predicates in the WHERE clause. This is most common and simplest in cases where all predicates are ANDed together so that their order can be shuffled without consequence.

Our results showed that testing for this kind of avoidance had no effect at all on the false positive rate for the exam dataset. For the coursework dataset, the increase is from 77.4% to 78.1% which is negligible.

Unfortunately, these results are inconclusive because very few of the questions posed for either dataset required more than one WHERE clause predicate.

5.4 All Variations

Combining all the variations of the shuffle algorithm gives a detection rate of 64.4% for the exam dataset and 80.6% for the coursework dataset. This is an increase of almost 8% in the false-positive rate for the exam dataset which is a cause for concern. If the equality algorithm becomes widespread and students try to avoid detection using the simple methods we have tested for, then automated detection may become infeasible because our results suggest that almost 2 in 3 submissions would be flagged as plagiarised even if no plagiarism took place. At best this requires the manual examination of a large majority of the submissions which defeats the purpose of automated detection. In the worst-case, students may be incorrectly accused of cheating.

6 VARIATION WITH ANSWER LENGTH

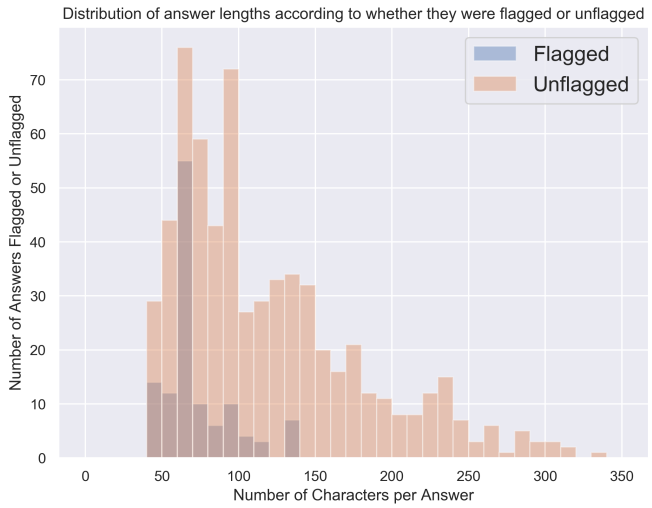
In the previous sections we have shown that the equality and shuffle algorithms result in a significant false-positive detection rate by finding many cases of plagiarism in a dataset where nothing was plagiarised. This would suggest that automated detection of plagiarism is not helpful for SQL assessments because the high false-positive rate requires manual inspection anyway.

However, as noted earlier, the length of an answer is likely to have a big impact on the success or failure of a detection algorithm. In this section, therefore, we consider the relationship between the length of answers and whether they are flagged as plagiarised. For simplicity, in this section we only consider the strict equality algorithm, the most relaxed equality algorithm (i.e. ignoring case, whitespace, quotes and brackets) and the shuffle algorithm combining all variants (i.e. removing alias names and shuffling operands and predicates).

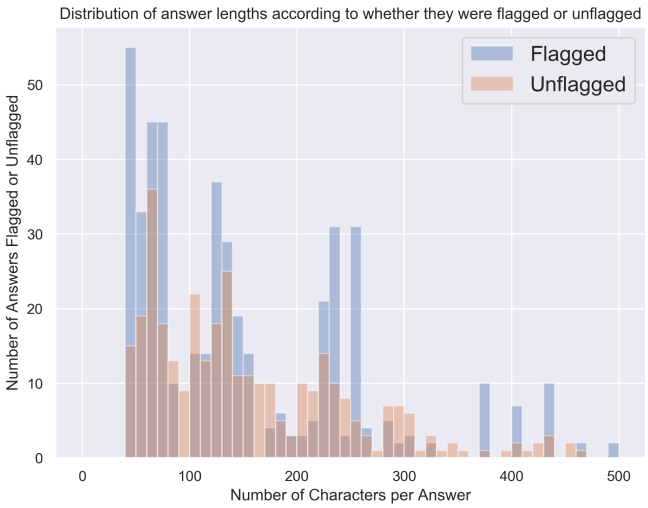
Figure 1 shows the distribution of the lengths of the answers for the two datasets split according to whether the answer was flagged as plagiarised or not. For the exam dataset (Figs. 1a, 1c, 1e) the pattern is that the flagged answers tend to be shorter and the longer answers tend to be unflagged. For the coursework dataset (Figs. 1b, 1d, 1f) there is no such difference in the distributions.

These results suggest that the high false-positive rate is only observed for shorter SQL queries. There is no definite cut-off point below which the algorithms are not effective and above which they are. However, in our datasets we can give approximate cut-off points. These points are different for the different algorithms.

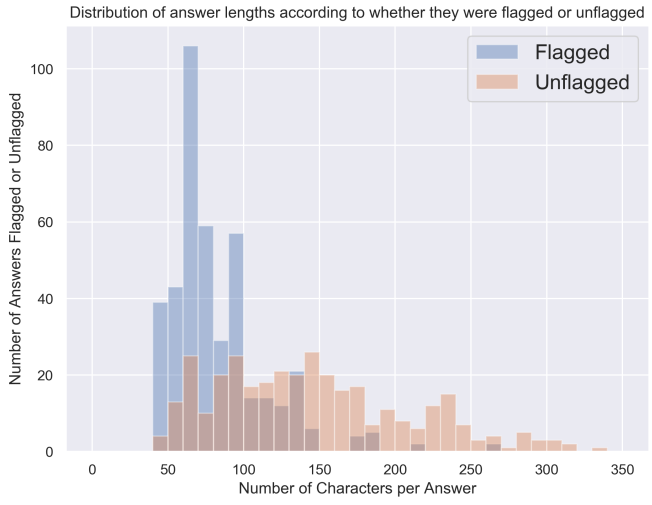
For the strict equality algorithm 120 characters appears to be a good choice. 23.1% of queries with 120 characters or fewer are flagged as plagiarised compared to just 2.7% of those with 120 or more. For the relaxed equality algorithm 200 characters offers a better cut-off point, with 60.2% of queries with under 200 characters flagged but only 5.4% of those with 200 or more.



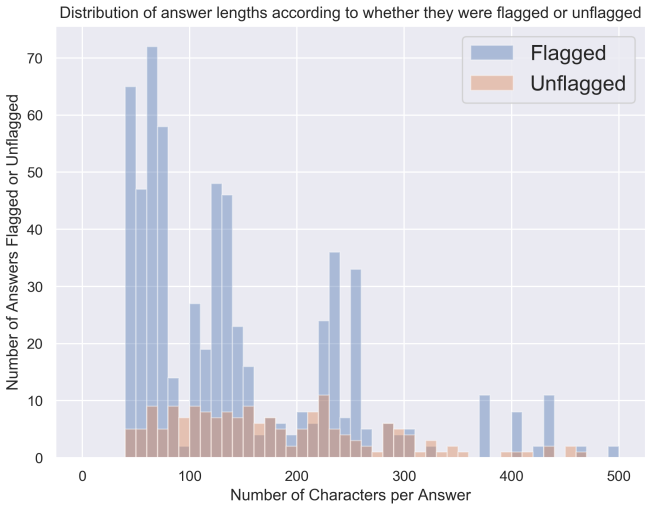
(a) Strict Equality Algorithm, Exam Dataset



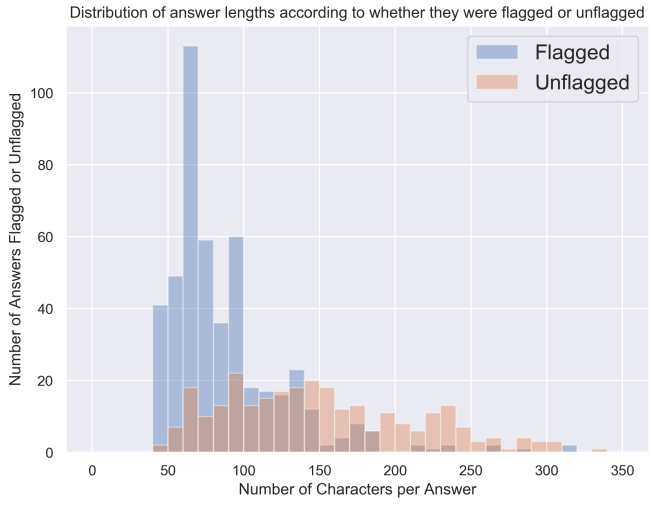
(b) Strict Equality Algorithm, Coursework Dataset



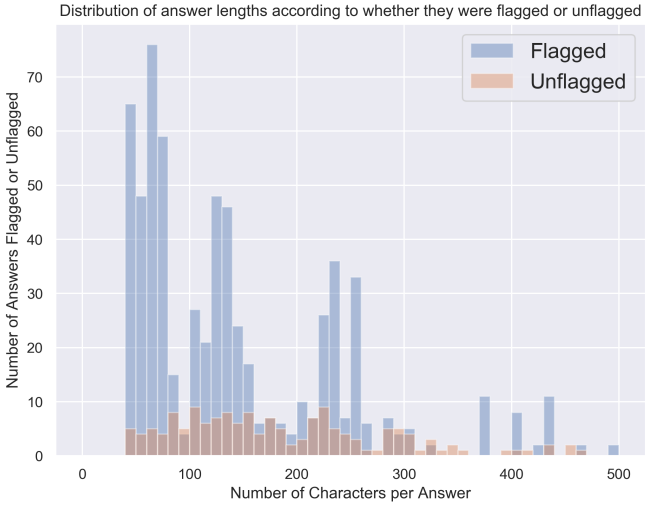
(c) Relaxed Equality Algorithm, Exam Dataset



(d) Relaxed Equality Algorithm, Coursework Dataset



(e) Shuffle Algorithm, Exam Dataset



(f) Shuffle Algorithm, Coursework Dataset

Figure 1: The distribution of answer lengths according to whether the answer was flagged as plagiarised or not.

For the shuffle algorithm the best cut-off point seems to also be 200 characters but with this algorithm there remains a relatively significant false positive rate even with longer queries. The rate for queries with 200 or fewer characters is 68.3% whereas for the longer queries it is 13.5%, which is not negligible.

In the coursework dataset there is less of a difference between the shorter and longer queries. Using a cut-off point of 300 characters, the detection rates are 58.1% and 55.4% for short and long queries respectively for the strict equality algorithm, 78.8% and 66.2% for the relaxed equality algorithm and 81.7% and 66.2% for the shuffle algorithm.

However, these results are not conclusive because in the exam dataset there are far more shorter queries than longer ones. The average length of each submission is 108 characters (s.d of 58) and only 9% of the submissions have 200 or more characters. Therefore, the lower detection rates observed for the longer queries may not be because the false-positive rates of those algorithms are lower for longer queries. It may simply be that there are far fewer other submissions for a submission to be incorrectly matched to. Therefore, further research is needed using other datasets to determine whether the false-positive rate does indeed fall with longer queries.

7 CONCLUSION

Automated assessment of SQL is becoming more widespread but automated detection of plagiarism in SQL has been only lightly studied. Automatically detecting plagiarism in SQL presents a unique challenge because the submissions tend to be much shorter and more rigidly structured than in imperative language assessments.

In this paper, we empirically examined the performance of various detection algorithms using two datasets with known plagiarism rates. Our results show that even with strict requirements for flagging submissions as plagiarised, there is a significant false-positive rate. With the absolutely strictest method (two submissions must match precisely as submitted), 15% of submissions are falsely flagged. Relaxing the criteria leads to more than 60% false positives.

In our datasets, this result only holds for the shorter of the SQL queries (e.g. under 120 characters for the strict algorithm and under 200 for the relaxed one). For longer queries, the false-positive rate is much lower.

These results suggest that automated detection of plagiarism is possible for SQL with longer queries. However, we note that our datasets tend towards shorter queries and therefore the low false-positive rate for longer queries may be due to a lack of other submissions to match to. Further research is needed with datasets containing longer queries.

As a concluding thought we note that no detection method can ever be 100% effective and catch every case of plagiarism. We believe this creates an ethical problem with using the detect-and-discipline approach to discourage plagiarism. Focusing on this approach to discouraging plagiarism leaves in place the temptation, and often the opportunity, to cheat. If relatively few cheaters are caught and punished, then plagiarism becomes increasingly attractive which is to the detriment of the cheaters as well as the non-cheaters. As Fraser has argued, this approach must be complemented with other approaches [5]. We suggest that more effort be directed towards designing plagiarism-proof assessments than designing detection algorithms. By removing the possibility of cheating, assessments become fairer and encourage more student learning.

REFERENCES

[1] Ilia Bider and David Rogers. 2016. YASQLT Yet another SQL tutor: A pragmatic approach. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 9975 LNCS. Springer, Cham, 197–206. https://doi.org/10.1007/978-3-319-47717-6_17

[2] Jess Bidgood and Jeremy B. Merrill. 2017. As Computer Coding Classes Swell, So Does Cheating. https://www.nytimes.com/2017/05/29/us/computer-science-cheating.htmlhttps://www.nytimes.com/2017/05/29/us/computer-science-cheating.html%0Ahttps://www.nytimes.com/2017/05/29/us/computer-science-cheating.html?smid=nytcore-ipad-share&smprod=nytcore-ipad&_r

[3] Samuel Breese, Evan Maicus, Matthew Peveler, and Barbara Cutler. 2018. Correlation of a Flexible Late Day Policy with Student Stress and Programming Assignment Plagiarism. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education - SIGCSE '18*. ACM Press, New York, New York, USA, 1089–1089. https://doi.org/10.1145/3159450.3162236

[4] Georgina Cosma and Mike Joy. 2006. Source-code plagiarism: A UK academic perspective. *Research Report No. 422* (2006), 1–74. https://www.dcs.warwick.ac.uk/report/pdfs/cs-rr-422.pdfhttp://eprints.dcs.warwick.ac.uk/52/

[5] Robert Fraser. 2014. Collaboration, collusion and plagiarism in computer science coursework. *Informatics in Education* 13, 2 (2014), 179–195. https://doi.org/10.15388/infedu.2014.01

[6] Amardeep Kahlon, Bonnie MacKellar, and Anastasia Kurdia. 2018. Combating the Wide Web of Plagiarism. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education - SIGCSE '18*. ACM Press, New York, New York, USA, 1069–1069. https://doi.org/10.1145/3159450.3162197

[7] Anthony Kleerekoper and Andrew Schofield. 2018. SQL tester: an online SQL assessment tool and its impact. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE 2018*. ACM Press, New York, New York, USA, 87–92. https://doi.org/10.1145/3197091.3197124

[8] Joshua License. 2017. testSQL: Learn SQL the Interactive Way. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '17*. ACM Press, New York, New York, USA, 376–376. https://doi.org/10.1145/3059009.3072991

[9] Tony Mason, Ada Gavrilovska, and David A. Joyner. 2019. Collaboration Versus Cheating. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education - SIGCSE '19*. ACM Press, New York, New York, USA, 1004–1010. https://doi.org/10.1145/3287324.3287443

[10] Matija Novak, Mike Joy, and Dragutin Kermek. 2019. Source-code Similarity Detection and Detection Tools Used in Academia. *ACM Transactions on Computing Education* 19, 3 (5 2019), 1–37. https://doi.org/10.1145/3313290

[11] Julia Opgen-Rhein, Bastian Küppers, and Ulrik Schroeder. 2018. An Application to Discover Cheating in Digital Exams. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*. ACM, 1–5. https://doi.org/10.1145/3279720.3279740

[12] Julia Prior. 2014. AsseSQL: an Online, Browser-based SQL Skills Assessment Tool. In *Proceedings of the 2014 conference on Innovation & technology in computer science education ITiCSE '14*. ACM Press, New York, New York, USA, 1. https://doi.org/10.1145/2591708.2602682

[13] Eric Roberts. 2002. Strategies for promoting academic integrity in CS courses. In *32nd Annual Frontiers in Education*, Vol. 2. IEEE, F3G–F3G.

[14] Gordon Russell and Andrew Cumming. 2005. Online assessment and checking of SQL: detecting and preventing plagiarism.. In *Teaching, Learning and Assessment in Databases*.

[15] Nikolai Scerbakov, Alexander Schukin, and Oleg Sabinin. 2018. Plagiarism detection in SQL student assignments. In *Advances in Intelligent Systems and Computing*, Vol. 716. Springer, Cham, 110–115. https://doi.org/10.1007/978-3-319-73204-6{_}14

[16] Judy Sheard, Simon, Matthew Butler, Katrina Falkner, Michael Morgan, and Amali Weerasinghe. 2017. Strategies for Maintaining Academic Integrity in First-Year Computing Courses. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE '17*. ACM Press, New York, New York, USA, 244–249. https://doi.org/10.1145/3059009.3059064

[17] Narjes Tahaei and David C. Noelle. 2018. Automated Plagiarism Detection for Computer Programming Exercises Based on Patterns of Resubmission. In *Proceedings of the 2018 ACM Conference on International Computing Education Research - ICER '18*. ACM Press, New York, New York, USA, 178–186. https://doi.org/10.1145/3230977.3231006